OpenRTM-aist (C++) - 機能 #2418

ECからRTCコールバック呼び出しを参照からサーバントにすることで効率化する

2012/05/07 09:00 - n-ando

ステータス:	終了	**開始日:**	2012/05/07
優先度:	通常	**期日:**	
担当者:		**進捗率:**	100%
カテゴリ:		**予定工数:**	0.00時間
対象バージョン:			

説明

現在のECはRTCをCORBA Object
referenceで保持しているが、これをサーバントに変換した上でコールバックを呼び出すように変更する。
こうすることで、onExecuteなど呼び出し時間が重要になるコールバック関数においてパフォーマンスを向上させる。

関係しているリビジョン

リビジョン 2355 - 2012/05/07 11:08 - n-ando

Now RTObjectStateMachine object calls RTC's callbacks via servant if it is available. refs #2418

リビジョン 2355 - 2012/05/07 11:08 - n-ando

Now RTObjectStateMachine object calls RTC's callbacks via servant if it is available. refs #2418

リビジョン 2355 - 2012/05/07 11:08 - n-ando

Now RTObjectStateMachine object calls RTC's callbacks via servant if it is available. refs #2418

履歴

#1 - 2012/05/07 09:01 - n-ando

OmniORBのプロセス内関数呼び出しについての調査

オブジェクト参照から実装クラスインスタンスへたどり着くのに結構な関数呼び出しを介していることがわかる。（数十から100程度）

#2 - 2012/05/07 09:06 - n-ando

実験

以下のコードを実行してみる。

```
interface Hoge
{
    void munya(in string msg, in short val);
};

//
// CORBA performance test program
//          Noriaki Ando, 2012.4.18
//
// This program compare between CORBA servant (implementation)
// operation call and CORBA object reference operation call.
//
// 1. Servant newed from Hoge_impl
// 2. Object reference obtained by _this()
// 3. Object reference from stringfied IOR
// 4. Servant obtained from object reference by reference_to_servant
//
// Servant.
// 0[sec], 3571[usec]
// Object reference from _this().
// 0[sec], 150894[usec]
// Created object from string IOR.
// 0[sec], 143678[usec]
// Servant obtained from reference_to_servant.
// 0[sec], 3543[usec]
// ORB termianted.
```

```
//
//  Servant  :      3.571  ns
//  Obj  ref  :  150.894  ns
//  IOR  ref  :  143.678  ns
//  Servant2:      3.543  ns
//  obj  ref  is  42  times  slower  than  servant
//
#include  <omniORB4/CORBA.h>
#include  <sys/time.h>
#include  <iostream>
#include  <pthread.h>

#include  "hoge.hh"
CORBA::ORB_var  orb;
class  Hoge_impl
    :  public  virtual  POA_Hoge
{
public:
    virtual  void  munya(const  char*  msg,  ::CORBA::Short  val)
    {
        //          std::cout  <<  msg  <<  std::endl;
        return;
    }
};
void*  orb_run(void*  arg)
{
    std::cout  <<  "orb  started"  <<  std::endl;
    orb->run();
    return  NULL;
}

#define  USEC_PAR_SEC  1000000
timeval  subtime(timeval  start,  timeval  end)
{
    timeval  ret;
    if  (end.tv_usec  <  start.tv_usec)
        {
            ret.tv_usec  =  end.tv_usec  +  USEC_PAR_SEC  -  start.tv_usec;
            ret.tv_sec  =  end.tv_sec  -  1  -  start.tv_sec;
            return  ret;
        }

    ret.tv_usec  =  end.tv_usec  -  start.tv_usec;
    ret.tv_sec  =  end.tv_sec  -  start.tv_sec;
    return  ret;
}

int  main(int  argc,  char**  argv)
{
      orb  =  CORBA::ORB_init(argc,  argv);
      CORBA::Object_var  obj  =  orb->resolve_initial_references((char*)"RootPOA");
      PortableServer::POA_var  poa  =  PortableServer::POA::_narrow(obj);
      PortableServer::POAManager_var  poamgr  =  poa->the_POAManager();
      poamgr->activate();

//        pthread_t  thread_handle;
//        pthread_attr_t  thread_attr;
//        pthread_create(&thread_handle,  NULL,  orb_run,  (void  *)  NULL);

      Hoge_impl*  hoge_servant  =  new  Hoge_impl();
      CORBA::Object_var  hoge_obj  =  hoge_servant->_this();
      Hoge_var  hoge_var  =  Hoge::_narrow(hoge_obj);
      std::string  ior;
      ior  =  orb->object_to_string(hoge_var);
      std::cout  <<  ior  <<  std::endl;

      CORBA::Object_var  newobj  =  orb->string_to_object(ior.c_str());
      Hoge_var  newhoge  =  Hoge::_narrow(newobj);
      Hoge_impl*  newservant
          =  dynamic_cast<Hoge_impl*>(poa->reference_to_servant(hoge_var));

      timeval  start,  end,  result;

      gettimeofday(&start,  NULL);
      for  (int  i(0);  i  <  1000000;  ++i)
```

```
        {
            hoge_servant->munya("hoge", 1000);
        }
    gettimeofday(&end, NULL);
    result = subtime(start, end);
    std::cout << "Servant." << std::endl;
    std::cout << result.tv_sec << "[sec], ";
    std::cout << result.tv_usec << "[usec]" << std::endl;

    gettimeofday(&start, NULL);
    for (int i(0); i < 1000000; ++i)
        {
            hoge_var->munya("hoge", 1000);
        }
    gettimeofday(&end, NULL);
    result = subtime(start, end);
    std::cout << "Object reference from _this()." << std::endl;
    std::cout << result.tv_sec << "[sec], ";
    std::cout << result.tv_usec << "[usec]" << std::endl;

    gettimeofday(&start, NULL);
    for (int i(0); i < 1000000; ++i)
        {
            newhoge->munya("hoge", 1000);
        }
    gettimeofday(&end, NULL);
    result = subtime(start, end);
    std::cout << "Created object from string IOR." << std::endl;
    std::cout << result.tv_sec << "[sec], ";
    std::cout << result.tv_usec << "[usec]" << std::endl;

    gettimeofday(&start, NULL);
    for (int i(0); i < 1000000; ++i)
        {
            newservant->munya("hoge", 1000);
        }
    gettimeofday(&end, NULL);
    result = subtime(start, end);
    std::cout << "Servant obtained from reference_to_servant." << std::endl;
    std::cout << result.tv_sec << "[sec], ";
    std::cout << result.tv_usec << "[usec]" << std::endl;

    //      orb->shutdown(true);
    //      pthread_join(thread_handle, NULL);
    //      std::cout << "ORB termianted." << std::endl;

    return 0;
}

all: hoge

OBJS = hogeSK.o hoge_impl.o
CFLAGS = -I.
LIBS = -lomniORB4 -lomnithread

hoge: $(OBJS)
        g++ -o hoge $(OBJS) $(LIBS)

hogeSK.o: hogeSK.cc
        g++ $(CFLAGS) -c -o hogeSK.o hogeSK.cc

hoge_impl.o: hoge_impl.cpp
        g++ $(CFLAGS) -c -o hoge_impl.o hoge_impl.cpp

hogeSK.cc: hoge.idl
        omniidl -bcxx hoge.idl

clean:
        rm -rf *.o hogeSK.cc hoge.hh hoge core *~
```

結果

Servant	3.571 ns	インスタンス化直後のサーバントへのポインタ経由呼び出し

Obj ref	150.894 ns	_this()で取得したオブジェクトリファレンス経由の呼び出し
IOR ref	143.678 ns	IOR文字列化後再度オブジェクト参照化したもの
Servant2	3.543 ns	IOR->オブジェクト参照->reference_to_servantで取得したサーバントのポインタ

大体40倍くらい違う。

#3 - 2012/05/07 11:06 - n-ando

RTObjectStateMachineを以下のように変更することで、オブジェクトリファレンス呼び出しをサーバント呼び出しにした。

```
Index: RTObjectStateMachine.cpp
===================================================================
--- RTObjectStateMachine.cpp (revision 2330)
+++ RTObjectStateMachine.cpp (working copy)
@@ -17,6 +17,8 @@
 */

 #include <rtm/RTObjectStateMachine.h>
+#include <rtm/Manager.h>
+#include <rtm/RTObject.h>
 #include <iostream>
 #include <stdio.h>

@@ -27,7 +29,8 @@
         : m_id(id),
           m_rtobj(RTC::LightweightRTObject::_duplicate(comp)),
           m_sm(NUM_OF_LIFECYCLESTATE),
-          m_ca(false), m_dfc(false), m_fsm(false), m_mode(false)
+          m_ca(false), m_dfc(false), m_fsm(false), m_mode(false),
+          m_rtobjPtr(NULL), m_measure(false)
     {
         m_caVar = RTC::ComponentAction::_nil();
         m_dfcVar = RTC::DataFlowComponentAction::_nil();
@@ -155,7 +158,11 @@
     setComponentAction(const RTC::LightweightRTObject_ptr comp)
     {
         m_caVar = RTC::ComponentAction::_narrow(comp);
-        if (!CORBA::is_nil(m_caVar)) { m_ca = true; }
+        if (CORBA::is_nil(m_caVar)) { return; }
+        m_ca = true;
+        PortableServer::POA_ptr poa = RTC::Manager::instance().getPOA();
+        m_rtobjPtr =
+          dynamic_cast<RTC::RTObject_impl*>(poa->reference_to_servant(comp));
     }

     void RTObjectStateMachine::
@@ -182,16 +189,40 @@
     // RTC::ComponentAction operations
     void RTObjectStateMachine::onStartup(void)
     {
+        // call Servant
+        if (m_rtobjPtr != NULL)
+          {
+            m_rtobjPtr->on_startup(m_id);
+            return;
+          }
+        // call Object reference
         if (!m_ca) { return; }
         m_caVar->on_startup(m_id);
     }
     void RTObjectStateMachine::onShutdown(void)
```

```
     {
+        //  call  Servant
+        if  (m_rtobjPtr  !=  NULL)
+          {
+            m_rtobjPtr->on_shutdown(m_id);
+            return;
+          }
+        //  call  Object  reference
          if  (!m_ca)  {  return;  }
          m_caVar->on_shutdown(m_id);
     }
     void  RTObjectStateMachine::onActivated(const  ExecContextStates&  st)
     {
+        //  call  Servant
+        if  (m_rtobjPtr  !=  NULL)
+          {
+            if  (m_rtobjPtr->on_activated(m_id)  !=  RTC::RTC_OK)
+              {
+                m_sm.goTo(RTC::ERROR_STATE);
+              }
+            return;
+          }
+        //  call  Object  reference
          if  (!m_ca)  {  return;  }
          if  (m_caVar->on_activated(m_id)  !=  RTC::RTC_OK)
           {
@@  -203,24  +234,55  @@

     void  RTObjectStateMachine::onDeactivated(const  ExecContextStates&  st)
     {
+        //  call  Servant
+        if  (m_rtobjPtr  !=  NULL)
+          {
+            m_rtobjPtr->on_deactivated(m_id);
+            return;
+          }
+        //  call  Object  reference
          if  (!m_ca)  {  return;  }
          m_caVar->on_deactivated(m_id);
     }

     void  RTObjectStateMachine::onAborting(const  ExecContextStates&  st)
     {
+        //  call  Servant
+        if  (m_rtobjPtr  !=  NULL)
+          {
+            m_rtobjPtr->on_aborting(m_id);
+            return;
+          }
+        //  call  Object  reference
          if  (!m_ca)  {  return;  }
-        m_caVar->on_error(m_id);
+        m_caVar->on_aborting(m_id);
     }

     void  RTObjectStateMachine::onError(const  ExecContextStates&  st)
     {
+        //  call  Servant
+        if  (m_rtobjPtr  !=  NULL)
+          {
+            m_rtobjPtr->on_error(m_id);
+            return;
+          }
+        //  call  Object  reference
          if  (!m_ca)  {  return;  }
          m_caVar->on_error(m_id);
     }

     void  RTObjectStateMachine::onReset(const  ExecContextStates&  st)
     {
+        //  call  Servant
+        if  (m_rtobjPtr  !=  NULL)
+          {
+            if  (m_rtobjPtr->on_reset(m_id)  !=  RTC::RTC_OK)
+              {
```

```
+                            m_sm.goTo(RTC::ERROR_STATE);
+                        }
+                    return;
+                }
+            // call Object reference
             if (!m_ca) { return; }
             if (m_caVar->on_reset(m_id) != RTC::RTC_OK)
                 {
@@ -233,9 +295,58 @@

       // RTC::DataflowComponentAction
       void RTObjectStateMachine::onExecute(const ExecContextStates& st)
       {
+          static int count;
+          double max_interval, min_interval, mean_interval, stddev;
+          // call Servant
+          if (m_rtobjPtr != NULL)
+              {
+                  if (m_measure) { m_svtMeasure.tick(); }
+                  if (m_rtobjPtr->on_execute(m_id) != RTC::RTC_OK)
+                      {
+                          m_sm.goTo(RTC::ERROR_STATE);
+                      }
+                  if (m_measure)
+                      {
+                          m_svtMeasure.tack();
+                          if (count > 1000)
+                              {
+                                  count = 0;
+                                  m_svtMeasure.getStatistics(max_interval, min_interval,
+                                                             mean_interval, stddev);
+                                  std::cout << "[servant] ";
+                                  std::cout << " max: " << max_interval;
+                                  std::cout << " min: " << min_interval;
+                                  std::cout << " mean: " << mean_interval;
+                                  std::cout << " stddev: " << stddev;
+                                  std::cout << std::endl;
+                              }
+                          ++count;
+                      }
+                  return;
+              }
+          // call Object reference
           if (!m_dfc) { return; }
-          if (m_dfcVar->on_execute(m_id) != RTC::RTC_OK)
+          if (m_measure) { m_refMeasure.tick(); }
+          RTC::ReturnCode_t ret = m_dfcVar->on_execute(m_id);
+          if (m_measure)
               {
+                  m_refMeasure.tack();
+                  if (count > 1000)
+                      {
+                          count = 0;
+                          m_refMeasure.getStatistics(max_interval, min_interval,
+                                                     mean_interval, stddev);
+                          std::cout << "[objref] ";
+                          std::cout << " max: " << max_interval;
+                          std::cout << " min: " << min_interval;
+                          std::cout << " mean: " << mean_interval;
+                          std::cout << " stddev: " << stddev;
+                          std::cout << std::endl;
+                      }
+                  ++count;
+              }
+          if (ret != RTC::RTC_OK)
+              {
                   m_sm.goTo(RTC::ERROR_STATE);
                   return;
               }
@@ -244,6 +355,16 @@

       void RTObjectStateMachine::onStateUpdate(const ExecContextStates& st)
       {
+          // call Servant
+          if (m_rtobjPtr != NULL)
```

```
+               {
+                   if  (m_rtobjPtr->on_state_update(m_id)  !=  RTC::RTC_OK)
+                   {
+                       m_sm.goTo(RTC::ERROR_STATE);
+                   }
+                   return;
+               }
+           //  call  Object  reference
             if  (!m_dfc)  {  return;  }
             if  (m_dfcVar->on_state_update(m_id)  !=  RTC::RTC_OK)
              {
@@  -255,6  +376,16  @@

       void  RTObjectStateMachine::onRateChanged(void)
       {
+           //  call  Servant
+           if  (m_rtobjPtr  !=  NULL)
+               {
+                   if  (m_rtobjPtr->on_rate_changed(m_id)  !=  RTC::RTC_OK)
+                   {
+                       m_sm.goTo(RTC::ERROR_STATE);
+                   }
+                   return;
+               }
+           //  call  Object  reference
             if  (!m_dfc)  {  return;  }
             if  (m_dfcVar->on_rate_changed(m_id)  !=  RTC::RTC_OK)
              {
```

#4 - 2012/05/07 11:16 - n-ando

- ステータス を 新規 から 解決 に変更

- 進捗率 を 0 から 100 に変更

以下の条件で実験を行った。

- ConfigSampleComp
  - EC: nowait
  - 表示なしに変更
  - log: NO, loglevel: NORMAL
  - 1000回ごとに統計

objref	17us
servant	2.5us

7倍くらい早くなったが、素のCORBA呼び出しの測定を行った前回の実験（40倍）に比べると、それほど変化がない。
また、呼び出し速度自体も、素のCORBAの場合、100nsオーダーだったか、今回はusオーダーであり、呼び出し時間も長い。

計測方法の問題のようなので、ひとまずECの改良自体は終了。

#5 - 2012/05/07 11:19 - n-ando

- ファイル 120419_newEC.zip を追加

#6 - 2012/05/07 11:19 - n-ando

- ステータス を 解決 から 終了 に変更

ファイル

120419_newEC.zip	1.5 MB	2012/05/07	n-ando